Evaluate a Siamese model: Ungraded Lecture Notebook

In [1]:
```
import trax.fastmath.numpy as np
```

```
INFO:tensorflow:tokens_length=568 inputs_length=512 targets_length=114 noise_density=0.15 mean_noise_span_length=3.0
```

Inspecting the necessary elements

In this lecture notebook you will learn how to evaluate a Siamese model using the accuracy metric. Because there are many steps before evaluating a Siamese network (as you will see in this week's assignment) the necessary elements and variables are replicated here using real data from the assignment:

- `q1` : vector with dimension `(batch_size X max_length)` containing first questions to compare in the test set.
- `q2` : vector with dimension `(batch_size X max_length)` containing second questions to compare in the test set.

  Notice that for each pair of vectors within a batch $([q1_1, q1_2, q1_3, \ldots], [q2_1, q2_2, q2_3, \ldots])$ $q1_i$ **is associated to** $q2_k$.

- `y_test` : 1 if $q1_i$ and $q2_k$ are duplicates, 0 otherwise.
- `v1` : output vector from the model's prediction associated with the first questions.
- `v2` : output vector from the model's prediction associated with the second questions.

You can inspect each one of these variables by running the following cells:

In [2]:
```
q1 = np.load('q1.npy')
print(f'q1 has shape: {q1.shape} \n\nAnd it looks like this: \n\n {q1}\n\n')
```

```
q1 has shape: (512, 64)

And it looks like this:

 [[ 32  38   4 ...   1   1   1]
 [ 30 156  78 ...   1   1   1]
 [ 32  38   4 ...   1   1   1]
 ...
 [ 32  33   4 ...   1   1   1]
 [ 30 156 317 ...   1   1   1]
 [ 30 156   6 ...   1   1   1]]
```

Notice those 1s on the right-hand side?

Hope you remember that the value of `1` was used for padding.

In [3]:
```
q2 = np.load('q2.npy')
print(f'q2 has shape: {q2.shape} \n\nAnd looks like this: \n\n {q2}\n\n')
```

```
q2 has shape: (512, 64)

And looks like this:

 [[   30   156    78 ...     1     1     1]
 [  283   156    78 ...     1     1     1]
 [   32    38     4 ...     1     1     1]
 ...
 [   32    33     4 ...     1     1     1]
 [   30   156    78 ...     1     1     1]
 [   30   156 10596 ...     1     1     1]]
```

In [4]:
```
y_test = np.load('y_test.npy')
print(f'y_test has shape: {y_test.shape} \n\nAnd looks like this: \n\n {y_test}\n\n')
```

```
y_test has shape: (512,)

And looks like this:

 [0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0
 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0
 0 0 0 1 0 0 1 1 0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0
 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0
 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1
 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1
 1 0 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 0
 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0
 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 0 1 0 1 1 1 0
 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 1 1 1
 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1
 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

In [5]:
```
v1 = np.load('v1.npy')
print(f'v1 has shape: {v1.shape} \n\nAnd looks like this: \n\n {v1}\n\n')
v2 = np.load('v2.npy')
print(f'v2 has shape: {v2.shape} \n\nAnd looks like this: \n\n {v2}\n\n')
```

```
v1 has shape: (512, 128)

And looks like this:

 [[ 0.01273625 -0.1496373  -0.01982759 ...  0.02205012 -0.00169148
  -0.01598107]
 [-0.05592084  0.05792497 -0.02226785 ...  0.08156938 -0.02570007
  -0.00503111]
 [ 0.05686752  0.0294889   0.04522024 ...  0.03141788 -0.08459651
  -0.00968536]
 ...
 [ 0.15115018  0.17791134  0.02200656 ... -0.00851707  0.00571415
  -0.00431194]
 [ 0.06995274  0.13110274  0.0202337  ... -0.00902792 -0.01221745
   0.00505962]
 [-0.16043712 -0.11899089 -0.15950686 ...  0.06544471 -0.01208312
  -0.01183368]]


v2 has shape: (512, 128)

And looks like this:

 [[ 0.07437647  0.02804951 -0.02974014 ...  0.02378932 -0.01696189
  -0.01897198]
 [ 0.03270066  0.15122835 -0.02175895 ...  0.00517202 -0.14617395
   0.00204823]
 [ 0.05635608  0.05454165  0.042222   ...  0.03831453 -0.05387777
  -0.01447786]
 ...
 [ 0.04727105 -0.06748016  0.04194937 ...  0.07600753 -0.03072828
   0.00400715]
 [ 0.00269269  0.15222628  0.01714724 ...  0.01482705 -0.0197884
   0.01389528]
 [-0.15475044 -0.15718803 -0.14732707 ...  0.04299919 -0.01070975
  -0.01318042]]
```

Calculating the accuracy

You will calculate the accuracy by iterating over the test set and checking if the model predicts right or wrong.

The first step is to set the accuracy to zero:

In [6]:
```python
accuracy = 0
```

You will also need the `batch size` and the `threshold` that determines if two questions are the same or not.

Note :A higher threshold means that only very similar questions will be considered as the same question.

In [7]:
```python
batch_size = 512 # Note: The max it can be is y_test.shape[0] i.e all the samples in test data
threshold = 0.7 # You can play around with threshold and then see the change in accuracy.
```

In the assignment you will iterate over multiple batches of data but since this is a simplified version only one batch is provided.

Note: Be careful with the indices when slicing the test data in the assignment!

The process is pretty straightforward:

- Iterate over each one of the elements in the batch
- Compute the cosine similarity between the predictions
    - For computing the cosine similarity, the two output vectors should have been normalized using L2 normalization meaning their magnitude will be 1. This has been taken care off by the Siamese network you will build in the assignment. Hence the cosine similarity here is just dot product between two vectors. You can check by implementing the usual cosine similarity formula and check if this holds or not.
- Determine if this value is greater than the threshold (If it is, consider the two questions as the same and return 1 else 0)
- Compare against the actual target and if the prediction matches, add 1 to the accuracy (increment the correct prediction counter)
- Divide the accuracy by the number of processed elements

In [8]:
```python
for j in range(batch_size):        # Iterate over each one of the elements in the batch

    d = np.dot(v1[j],v2[j])        # Compute the cosine similarity between the predictions as l2 normalized, ||v1[j]||==||v2[
    res = d > threshold            # Determine if this value is greater than the threshold (if it is consider the two questio
    accuracy += (y_test[j] == res) # Compare against the actual target and if the prediction matches, add 1 to the accuracy

accuracy = accuracy / batch_size   # Divide the accuracy by the number of processed elements
```

In [9]:
```python
print(f'The accuracy of the model is: {accuracy}')
```

```
The accuracy of the model is: 0.7421875
```

Congratulations on finishing this lecture notebook!

Now you should have a clearer understanding of how to evaluate your Siamese language models using the accuracy metric.

Keep it up!